Projet ISN : Puzzle Image

Une application Web de jeu interactif

14 septembre 2013

Ce document propose un projet pour les élèves de Terminale S inscrits en spécialité Informatique et Sciences du Numérique.

Les langages HTML, CSS et JavaScrip sont brièvement présentés et seuls les éléments de ces langages nécessaires à la réalisation de ce projet y sont décrits plus en détails.

Table des matières

1 Puzzle Image **3**

2 Projet **3**

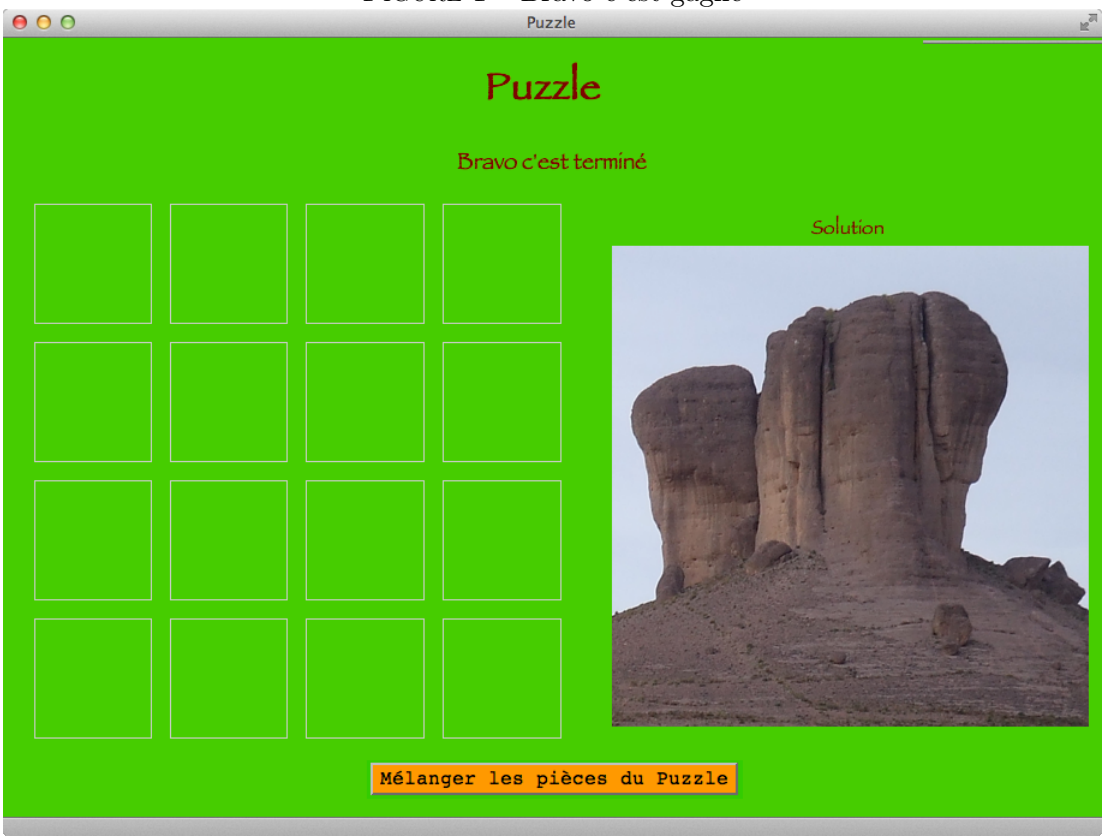
3 Plan de travail **4**
- 3.1 Les outils nécessaires . . . 4
- 3.2 Les fichiers à créer . . . 4

4 Création des fichiers du projet **5**
- 4.1 `"index.html"` une page Web codée en HTML . . . 5
  - 4.1.1 Le langage HTML . . . 5
  - 4.1.2 En-tête du document . . . 5
  - 4.1.3 Corps du document . . . 6
  - 4.1.4 Le DOM (*Document Object Model*) . . . 9
- 4.2 `"style.css"` une feuille de styles codée en CSS . . . 10
  - 4.2.1 Le langage CSS . . . 10
  - 4.2.2 Balises HTML, objets du DOM, widgets graphiques : . . . 11
  - 4.2.3 Déclaration de propriété : . . . 12
  - 4.2.4 Définition des styles . . . 15
- 4.3 `"code.js"` un programme codé en JavaScript . . . 16
  - 4.3.1 Le langage JavaScript . . . 16
  - 4.3.2 Initialisation du programme . . . 17
  - 4.3.3 Débug : Alerte JavaScript . . . 17
  - 4.3.4 Modification d'un objet du DOM . . . 18
  - 4.3.5 Action du bouton . . . 18
  - 4.3.6 Modélisation du jeu . . . 18
  - 4.3.7 Que faire de plus ? . . . 21

Table des figures

1 Bravo c'est gagné 2
2 Page d'accueil du Puzzle 3
3 Fenêtre associée au DOM 10
4 Rendus obtenus sans style ou avec style 11
5 Structure d'un widget graphique 12

FIGURE 1 – Bravo c'est gagné



1 Puzzle Image

Le jeu se présente comme une image découpée en plusieurs morceaux. Ces morceaux sont mélangés, et il faut le remettre en ordre pour reconstituer l'image.

On pourra, par exemple, prendre une image carrée de 400×400 pixels découpée en 16 morceaux 100×100 pixels.

Les morceaux seront donnés, à gauche, dans le désordre et il faudra les réordonner pour reconstituer l'image, à droite.

2 Projet

On demande de réaliser une application Web interactive qui permet de jouer avec ce puzzle.

FIGURE 2 – Page d'accueil du Puzzle

Puzzle
Puzzle
Clic sur la pièce sélectionnée puis Clic sur la zone vide où la placer
Solution
Mélanger les pièces du Puzzle

Cette application sera constituée de 3 fichiers :

1. une page HTML pour visualiser le puzzle
2. une feuille de style CSS pour enjoliver l'affichage
3. un programme JavaScript pour simuler le jeu et gérer les évènements

On trouvera des informations utiles dans la rubrique « Page Web » du site ISN, « Spécialité Informatique et Sciences du Numérique » :

http://mathmac.free.fr/-PWAccueil.html

3 Plan de travail

Une application Web se présente comme un dossier qui peut être téléversé sur un serveur distant contenant un site Web. Cependant, cette application peut fonctionner et être mise au point, en local, sur un ordinateur personnel.

Le dossier à créer contiendra au minimum trois fichiers de textes codés selon des formats différents et que l'on va rédiger successivement.

3.1 Les outils nécessaires

Pour réaliser ce projet et créer ces fichiers, il faut utiliser :

— Un éditeur de texte brut, c'est à dire sans option de mise en page.
Il est cependant préférable d'utiliser un éditeur prévu pour la programmation de code et qui offre au moins la colorisation syntaxique.

— Un navigateur Internet dans lequel on ouvre le projet pour en contrôler le fonctionnement.
La plupart de navigateurs Internet disposent de divers outils de développement qui peuvent se révéler utiles lors de la mise au point du projet.

Comme navigateur Internet, on conseille d'utiliser `Firefox` :

http://www.mozilla.org/fr/firefox/

Pour vérifier les codes et les programmes créés, on peut ajouter à `Firefox` le module `Firebug` qui fournit des outils de développement :

http://www.getfirebug.com/

Ces outils sont indispensables pour la mise au point d'une application Web. En effet, pour ne pas interrompre trop souvent la navigation sur Internet, les navigateurs sont extrêmement robustes. Ils ne signalent pas la plupart de erreurs provoquées par des pages HTML incorrectes ou par les bugs dans les programmes. Le programme ne se bloque pas, même s'il peut faire n'importe quoi.

Il faut donc contrôler attentivement son travail l'aide de ces outils de développement qui proposent entre autres une console d'erreurs.

3.2 Les fichiers à créer

Il y a trois fichiers de texte à créer dans un même dossier :

1. Le fichier `"index.html"` codé en HTML contient la description de la page Web affichée dans le navigateur.
   Cette page devra présenter au moins :
   — un titre
   — un message de mode d'emploi ou de compte rendu de jeu
   — une grille de jeu interactive
   — un bouton permettant de lancer une nouvelle partie
   Voir paragraphe 4.1 page 5.
2. Le fichier `"style.css"` codé en CSS contient les déclarations des styles des différents éléments affichés dans la page Web.
   Voir paragraphe 4.2 page 10.
3. Le fichier `"code.js"` codé en JavaScript contient le programme qui gère le fonctionnement de l'application Web interactive.

Voir paragraphe 4.3 page 16.

4 Création des fichiers du projet

4.1 "index.html" une page Web codée en HTML

On commence par créer un fichier de texte au format HTML, qui sera la page d'entrée de l'application. Comme il est d'usage sur le Web, on nomme ce fichier : "index.html". Si on ouvre ce fichier dans un navigateur Internet on verra la page Web que l'on désire publier.

4.1.1 Le langage HTML

Le langage HTML (*HyperText Markup Language*) est un langage qui permet de structurer un texte ordinaire à l'aide de balises (*tag* en anglais).

Cette section propose une initiation au langage HTML limitée à ce qui est utile pour le projet. On trouvera de nombreuses informations sur Internet.

Dans la rubrique « Page Web » du site ISN une application Web interactive permet d'effectuer des tests simples sur le langage HTML :

http://mathmac.free.fr/PageWeb/AppliHTML/index.html

L'ensemble du code est compris entre <html> la balise qui marque le début du document et </html> la balise de fin du document.

Un document est constitué d'un en-tête placé entre les balises <head> et </head> suivie d'un corps de texte placé entre les balises <body> et </body>.

Le code HTML du document a donc la forme suivante :

```
<html>
  <head>
    ... En-tête du document ...
  </head>
  <body>
    ... Corps du document ...
  </body>
</html>
```

On précise page 5 quelles indications techniques fournir dans l'en-tête du document pour afficher correctement la page Web.

On présente page 6 les balises HTML utiles pour le codage du corps du document qui définit la page Web.

4.1.2 En-tête du document

L'en-tête du document contient des informations techniques concernant le mode d'affichage du texte de la page Web dans la fenêtre du navigateur.

Les balises <meta> permettent de donner des indications au navigateur pour lui permettre d'afficher correctement la page proposée.

Il est fortement conseillé de placer dans l'en-tête, au moins une balise <meta> pour préciser le mode d'encodage du texte, ce qui est important pour que les caractères accentués soient correctement affichés dans le navigateur.

Cette balise doit contenir ce qu'on appelle une *liste attribut-valeur* .

Une liste attribut-valeur peut figurer dans toute balise HTML. C'est un texte inséré entre le nom de la balise et le signe `>` qui la termine. Un *couple attribut-valeur* est un nom d'attribut suivi du signe `=` suivi d'une valeur d'attribut écrite entre guillemets doubles `"`. Les différents couples attribut-valeur doivent être séparés par des espaces.

Dans notre projet, `http-equiv="content-type"` sera le premier couple attribut-valeur de notre balise `<meta>`, il précise que cette balise concerne le contenu du document.

Un second couple attribut-valeur précise le type et l'encodage du contenu du document, par exemple : `content="text/html; charset=iso-8859-1"`

Le format iso-8859-1 aussi nommé iso-latin-1 est un encodage usuel de fichier texte. Dans ce cas on placera dans l'en-tête le code suivant :

```
<meta http-equiv="content-type" content="text/html; charset=iso-8859-1">
```

Mais, si on dispose d'un éditeur de texte qui le permet, on préfère utiliser l'encodage UTF-8, plus moderne et qui donne accès à tous les caractères Unicode. On placera donc alors dans l'en-tête le code suivant :

```
<meta http-equiv="content-type" content="text/html; charset=utf-8">
```

On peut aussi définir dans l'en-tête le texte qui est affiché dans la barre de titre de la fenêtre du navigateur à l'aide des balises `<title>` et `</title>`

Le code HTML de l'en-tête du document peut ainsi prendre la forme suivante :

```
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8">
    <title>Puzzle</title>
  </head>
  <body>
    ... Corps du document ...
  </body>
</html>
```

À partir de maintenant, le code étant ainsi rédigé, il est fortement conseillé d'ouvrir le fichier `"index.html"` dans un navigateur Internet pour voir ce qui est affiché, puis de recommencer de même à la suite de chaque nouvelle modification apporté au code, pour contrôler si tout se passe comme prévu.

Attention : il est indispensable de vérifier en permanence le travail effectué. Pour cela on enregistre les fichiers modifiés et on ouvre le fichier `"index.html"` dans le navigateur Internet pour contrôler le rendu.

4.1.3 Corps du document

Le corps du document contient tout ce que l'on désire afficher dans notre page Web, titres, textes, images, boutons, etc.

On décrira uniquement dans ce paragraphe les balises HTML utiles à la réalisation du projet. Les éléments indispensables au puzzle seront d'abord définis simplement, puis leur style graphique sera amélioré ensuite comme on le proposera au paragraphe 4.2 page 10.

Mise en page du texte : Tout le texte placé dans le corps d'une page HTML est affiché sur une seule ligne en flot continu, c'est à dire sans tenir compte des espaces en surnombre ou des sauts de lignes présents dans le code. Les seuls retours à la ligne sont ceux qui sont effectués automatiquement quand le bord droit de la fenêtre est atteint.

Quelques balises pré-définies permettent cependant un peu mise en page.

Il est possible de fractionner la page du document en *paragraphes* en plaçant les différents textes entre les balises `<p>` et `</p>`

Il est aussi possible de fractionner la page en *divisions* en écrivant entre les balises `<div>` et `</div>`. Une division peut contenir, non seulement du texte, mais aussi d'autres éléments graphiques comme les listes, les tables, les images, les boutons, etc.

Il y a aussi des balises pré-définies pour afficher six niveaux de *titres* :

— `<h1>`section`</h1>`
— `<h2>`sous-section`</2>`
— ... et ainsi de suite jusqu'à ...
— `<h6>...</h6>`

Le navigateur attribue à tous ces textes placés entre des balises, des styles d'affichage particuliers pré-définis.

On verra plus loin, au paragraphe 4.2 page 10, que le mode d'affichage, ou *rendu*, du code HTML peut toutefois être modifié à l'aide d'une feuille de style.

L'application Web disponible sur le site ISN et déjà signalée page 5, permet de tester rapidement le rendu proposé par défaut par votre navigateur Internet pour les différentes balises HTML :

http://mathmac.free.fr/PageWeb/AppliHTML/index.html

On peut commencer à coder la page du projet :

— écrire le titre
— écrire un message de mode d'emploi ou de compte rendu de jeu

Ne pas oublier de tester à chaque étape.

Il reste maintenant à placer les éléments du jeu, c'est à dire :

— une grille de jeu de 5 lignes de 5 colonnes
— un bouton permettant de lancer une nouvelle partie

La balise `<table>` : Elle permet d'aligner des textes sur une grille structurée en lignes et colonnes.

Entre les balises `<table>` et `</table>`, chaque ligne est définie ente des balises `<tr>` et `</tr>`, et dans chaque ligne les colonnes sont définies entre des balises `<td>` et `</td>`.

Par exemple le code HTML suivant :

```
<table>
  <tr>
    <td>a</td>  <td>bbbb</td>
  </tr>
  <tr>
    <td>AAAAA</td>  <td>B</td>
  </tr>
</table>
```

ce code va afficher :

a	bbbb
AAAAA	B

On peut ajouter dans la balise `<table>` une liste attributs-valeurs qui permet de modifier le rendu de l'affichge.

L'attribut `align` permet de fixer l'alignement horizontal de la table dans la fenêtre du navigateur. Il peut prendre les valeurs : `"left"` qui est la valeur par défaut, `"right"` ou `"center"`.

Les principaux attributs pré-définis de mise en forme sont les suivants :

— `border` définit l'épaisseur du trait qui encadre les cellules
— `cellspacing` définit l'espace entre les cellules de la grille
— `cellpadding` définit l'espace entre le cadre de la cellule et le texte

La valeur de ces attributs est un entier, entre guillemets, qui représente un nombre de pixels.

Par exemple :

```
<table border="1" cellspacing="5" cellpadding="2">
  <tr>
    <td>Ligne 1,   Colonne 1</td>
    <td>Ligne 1, Colonne 2</td>
  </tr>
  <tr>
    <td>  Ligne 2, Colonne 1  </td>
    <td>Ligne 2, Colonne 2</td>
  </tr>
</table>
```

ce code va afficher :

Ligne 1, Colonne 1	Ligne 1, Colonne 2
Ligne 2, Colonne 1	Ligne 2, Colonne 2

Pour notre Puzzle, on peut par exemple découper une image carrée de 400×400 pixels en 16 pièces carrées de 100×100 pixels. Ainsi la solution sera affichée sur une grille 4×4. Les pièces du Puzzle, mélangées au départ, seront placées sur une autre grille 4×4.

Pour obtenir l'aspect visuel présenté Figure 2 page 3, il est possible de placer d'aligner deux tables en les plaçant dans les cellules d'une autre table, comme sur ce modèle :

```
<table cellspacing="10">
  <tr>
    <td><table>...</table></td>
    <td><table>...</table></td>
  </tr>
</table>
```

La balise `<img />` : Cette balise permet d'inséré un image dans un document HTML. Pour cela il faut préciser la valeur de certains attributs pré-définis :

— `src` définit le nom du fichier image à utiliser
— `alt` définit un texte affiché en cas d'absence du du fichier image
— `width` définit la largeur de l'image affichée (optionnel)
— `height` définit la hauteur de l'image affichée (optionnel)

Exemple :

```
<img src="monImage.png" alt="Mon image">
```

Ici, le fichier `"monImage.png"` doit être disponible dans le même dossier que celui qui contient `"index.html"`. En cas d'absence du fichier, le texte `"Mon image"` sera affiché. Comme les attributs `width` et `height` ne sont pas spécifiés, l'image sera affichée dans sa taille originale.

Remarque : Notre Puzzle exige le découpage d'une image.

Ce découpage, assez délicat, peut être effectué à l'aide d'un logiciel de dessin spécialisé.

Cependant il y a une autre méthode qui permet d'éviter ce découpage en utilesant le langage CSS comme on l'expliquera en détail au paragraphe 4.2.3 page 13.

La balise `<button>` : On veut que l'utilisateur dispose d'un bouton sur lequel cliquer pour mélanger les pièces du Puzzle et jouer une nouvelle partie.

Le texte figurant dans le bouton sera écrit entre les balises `<button>` et `</button>`.

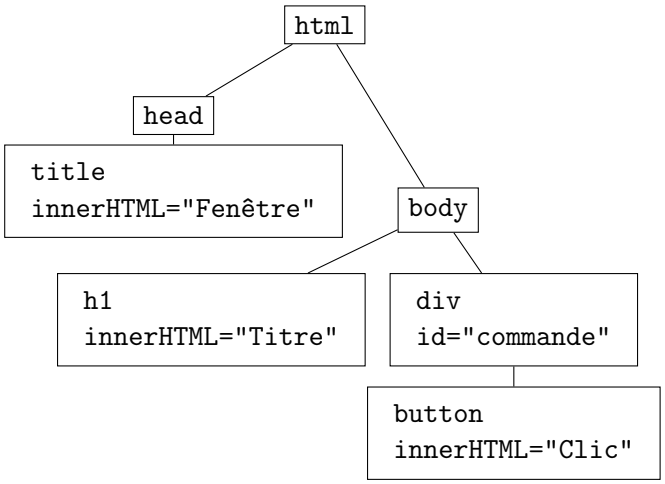
Pour modifier le style du bouton, voir paragraphe 4.2 page 10. Pour définir l'action à exécuter lorsque l'utilisateur effectue un clic sur le bouton, on se reportera au paragraphe 4.3.5 page 18.

4.1.4 Le DOM (*Document Object Model*)

Remarque : Le modèle du DOM présenté dans ce document est simplifié. Il s'agit du modèle contrôlé par le langage JavaScript.

Le code html affiché ci-dessous, à gauche, peut être représenté schématiquement par un arbre sous la forme affichée à droite :

```
<html>
<head>
  <title>Fenêtre</title>
</head>
<body>
  <h1>Titre</h1>
  <div id="commande">
    <button>Clic</button>
  </div>
</body>
</html>
```



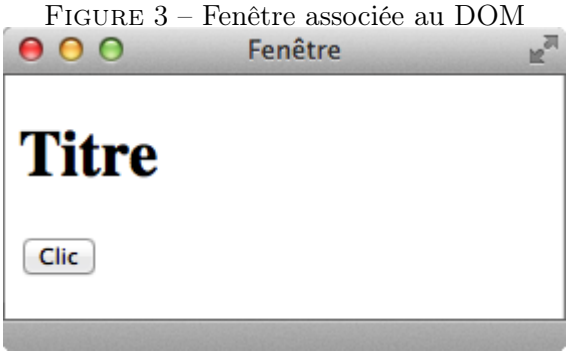
Cet arbre constitue ce qu'on appelle le *Modèle Objet du Document*, en anglais *Document Object Model*, et en abrégé, le DOM.

Le rendu associé au DOM et affiché par le navigateur est visible Figure 3.

En résumé : À partir de chaque balise du document HTML est créé un objet JavaScript associé au DOM avec sa liste de propriétés. Chaque attribut de balise correspond à une propriété de l'objet et le texte contenu entre les deux balises est la valeur de la propriété `innerHTML`. Ensuite, à partir de chaque objet du DOM, un élément graphique (*widget*) est affiché dans la fenêtre du navigateur. Ainsi dans la suite de cette documentation, quand on fait référence à un élément du DOM, on considère indifféremment l'objet du DOM, la balise et son contenu ou le widget graphique qui le représente.

Nous avons vu dans ce paragraphe 4.1 comment définir la page HTML. On verra dans le paragraphe suivant 4.2 page 10, comment modifier le style de l'affichage, et ensuite, au

FIGURE 3 – Fenêtre associée au DOM



paragraphe 4.3 page 16, comment réagir aux interactions de l'utilisateur et animer la page Web.

4.2 `"style.css"` une feuille de styles codée en CSS

On va maintenant créer un nouveau fichier de texte que l'on va nommer, par exemple `"style.css"`. Il sera codé au format CSS.

Le langage CSS (*Cascading Style Sheets*) est un langage qui permet de déclarer le style d'affichage des divers éléments d'une page HTML.

On travaillera donc sur deux fichiers en parallèle, `"index.html"` décrit au paragraphe précédent et `"style.css"`.

Avant de commencer la rédaction de notre feuille de style, on va d'abord lier notre page HTML, `"index.html"`, à notre feuille de style, `"style.css"`.

Pour cela on va ajouter dans l'en-tête de notre document HTML une balise `<link>` *(lien)* avec sa liste attributs-valeurs sous la forme :

```
<link rel="stylesheet" type="text/css" href="style.css">
```

Les différents attributs sont :

— `rel="stylesheet"` indique qu'il s'agit d'une feuille de style
— `type="text/css"` précise le type de codage
— `href="style.css"` donne le nom du fichier de la feuille de style

Maintenant, à l'ouverture du fichier `"index.html"` dans un navigateur Internet le fichier auxiliaire `"style.css"` est aussi automatiquement ouvert et interprété.

4.2.1 Le langage CSS

Une feuille de style CSS est un fichier texte qui contient des *règles de déclarations de style* à appliquer aux éléments d'une page Web.

Chaque règle de déclarations de style est composée d'un sélecteur suivi d'une liste de déclarations de propriétés entre accolades, sous la forme :

sélecteur { *propriété* : *valeur* ; *propriété* : *valeur* ; ... }

Sélecteur : Le sélecteur indique sur quels objets du DOM va porter le style défini. Il y a trois types de sélecteurs :

— `tag` un simple nom de balise ou tag
— `#identificateur` un nom précédé de `#`

— `.classe` un nom précédé de `.` le point

Chaque sélecteur fournit une liste d'un ou plusieurs objets sélectionnés dans le DOM.

Si le sélecteur est est un nom de balise ou tag, le style s'applique au contenu de tous les objets du DOM associés aux balises portant ce nom.

Par exemple : la règle `p { color: red }` provoque l'affichage en rouge de tous les paragraphes, c'est à dire les textes écrits entre `<p>` et `</p>`.

Si le sélecteur est un nom précédé de `#` le style s'applique au contenu de l'unique objet du DOM associé à la balise, quelque soit son tag, dont l'attribut `id` a pour valeur le nom préfixé par le `#`. Cet identificateur ne doit figurer que dans une unique balise de la page HTML.

Par exemple : si on a la règle `#vert { background-color: green }` et si, dans le code HTML, on a une balise identifiée par `"vert"`, son contenu sera affiché sur fond vert.

Comme cette division par exemple : `<div id="vert"> ... </div>`

Si le sélecteur est un nom précédé de `.` le style s'applique au contenu de touts les objets du DOM associés aux balises, quelque que soit leurs tags, dont l'attribut `class` a pour valeur le nom préfixé par le point.

Par exemple : si on a la règle `.gras { font-weight: bold }` le texte contenu dans toutes les balises de classe `"gras"`, quelque soit les tags, sera affiché en gras.

Comme les balises suivantes :

`<p class="gras"> ... </p>` ou `<table class="gras"> ... </table>`

4.2.2 Balises HTML, objets du DOM, widgets graphiques :

Revenons sur l'exemple qui présente le DOM affiché Figure 3. Dans cet exemple, le navigateur utilise un style par défaut.

Comment peut-on modifier ce style pour obtenir le rendu de droite de la Figure 4 ?

FIGURE 4 – Rendus obtenus sans style ou avec style

Il suffit d'écrire ce code dans la feuille de style :

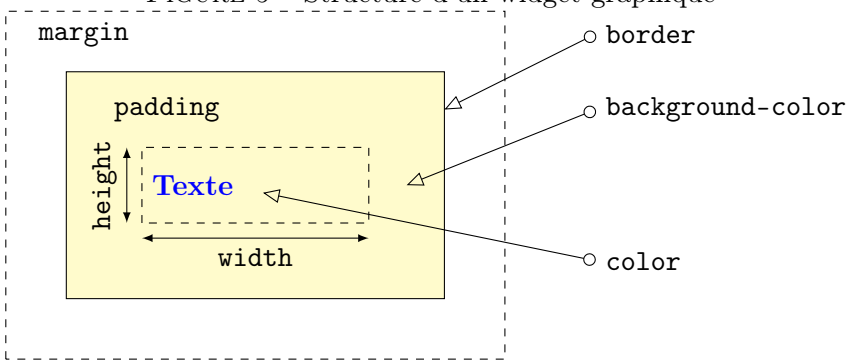
```
h1 {
   margin: 10px 25px 10px 25px;
   border: 5px blue solid;
   padding: 20px;
   background-color: yellow;
   text-align: center;
   color: maroon;
}
```

Par défaut, le navigateur affiche dans la fenêtre les widgets, comme un flot continu (de gauche à droite et de haut en bas) de rectangles de différentes tailles. Sur la Figure 5 on voit

un widget contenant le mot "Texte" et sur lequel sont notées les différentes propriétés qui permettent de préciser les dimensions de son rectangle.

Cette figure permet de comprendre la règle de déclaration de style présentée ci-dessus.

FIGURE 5 – Structure d'un widget graphique



Chaque navigateur donne à toutes ces propriétés des valeurs par défaut qu'il est possible de modifier dans la feuille de style.

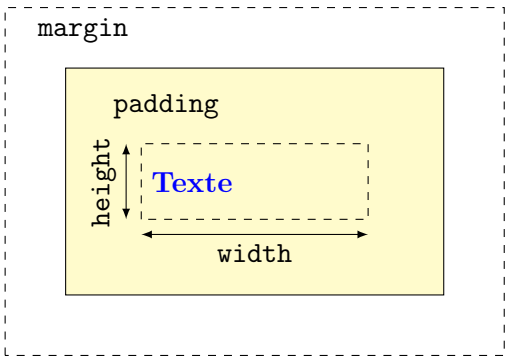
Pour bien comprendre cet exemple, voici maintenant une description des déclarations de propriétés usuelles.

4.2.3 Déclaration de propriété :

Les noms des propriétés sont des noms prédéfinis du langage CSS.

On va donc décrire les propriétés CSS usuelles et les valeurs qu'elles peuvent prendre.

Propriétés géométriques : Elles permettent de fixer les dimensions des widgets selon le modèle présenté Figure 5 et repris ci-dessous. Par défaut, le rectangle du widget adapte ses dimensions pour se coller autour du texte qu'il encadre. On peut cependant modifier tous les paramètres :



— `width` largeur (unités : `mm`, `cm`, `px` *pixels*)
exemple : `width: 55mm;`

— `height` hauteur
exemple : `height: 25px;`

— `padding` espace entre le texte et la bordure (unités : `mm`, `cm`, `px` et `%`)
exemple : `padding: 20px;`

— `margin` espace autour de la bordure
exemple : `margin: 10px 25% 10px 25%;`

Pour `padding` et `margin` il y a deux formes possibles. Si une seule mesure est fournie elle s'applique aux 4 côtés de rectangle.

Mais, il est possible de donner 4 valeurs distinctes qui représentent successivement les espaces (*top right bottom left*) au dessus, à droite, au dessous et à gauche.

Si on utilise `%` comme unité cela représente un pourcentage de la largeur de la fenêtre, ce qui significatif uniquement à gauche et à droite.

Propriétés typographiques : Elles permettent de modifier le mode d'écriture et l'aspect des textes :

- `text-align` alignement horizontal du texte dans le rectangle, cette propriété concerne les balises `<div>` (valeurs : `center`, `left`, `right`)
  exemple : `text-align: center;`
- `font-style` style typographique du texte (valeurs : `normal`, `italic`)
  exemple : `font-style: italic;`
- `font-weight` *graisse* (épaisseur typographique) des caractères du texte (valeurs : `normal`, `bold`, `bolder` et `lighter`)
  exemple : `font-weight: bold;`
- `font-size` taille des caractères du texte (valeurs : `xx-small`, `x-small`, `smaller`, `small`, `medium`, `large`, `larger`, `x-large` et `xx-large`)
  exemple : `font-size: x-large;`
- `font-family` famille des caractères du texte (valeurs : `serif`, `sans-serif`, `monospace` et `fantasy`)
  exemple : `font-family: monospace;`

Propriétés décoratives : Elles permettent de définir des encadrement ou de modifier les couleurs des widgets.

- `color` couleur du texte (couleur : voir ci-dessous)
  exemple : `color: red;`
- `background-color` couleur du fond du widget
  exemple : `background-color: yellow;`
- `border` bordure du widget
  exemple : `border: 1px blue solid;`

Les valeurs de couleurs sont : `aqua`, `black`, `blue`, `fushia`, `gray`, `green`, `lime`, `marron`, `navy`, `olive`, `purple`, `red`, `silver`, `teal`, `white`, `yellow`, mais on peut aussi définir la couleur par son code RVB hexadécimal. On écrit le caractère `#` suivi de 3 chiffres hexa (`0-F`) pour Rouge, Vert, et Bleu.

Par exemple : rouge `#F00`, vert `#0F0`, bleu `#00F`, blanc `#FFF`, noir `#000`.

Pour la bordure, la propriété `border` est suivie de trois paramètres : l'épaisseur du trait, la couleur du trait et le type de trait choisi parmi les valeurs suivantes : `dotted` (*pointillé*), `dashed` (*tireté*) ou `solid` (*plein*).

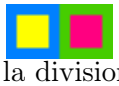
Compléments sur les propriétés du fond (`background`) : Nous allons présenter ici quelques astuces permettant d'afficher des images découpées comme l'exige notre Puzzle.

Nous avons vu ci-dessus qu'il est possible d'affecter une couleur au fond d'un widget. Il est aussi possible d'afficher une image sur le fond d'un widget. Voici les propriétés qui permettent de contrôler cet affichage sur le fond :

— `background-image` image du fond du widget (valeur : `url(`*`Nom du fichier image`*`)`)
exemple : `background-image: url(monImage.png)` ;
— `background-repeat` mode d'affichage (valeurs : `no-repeat`, `repeat-x`, `repeat-y`)
exemple : `background-repeat: no-repeat;`
— `background-position` position de l'image dans le fond (valeur : coordonnées `top left`)
exemple : `background-position: 0px -10px;`

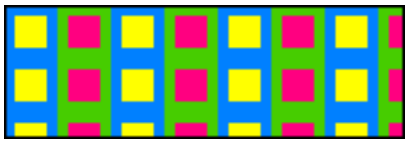
Le fichier image doit être dans le dossier contenta les autres fichiers. L'image est affichée, par défaut, de façon à remplir complètement le fond du widget concerné à partir du coin en haut à gauche (`top left`). Si l'image est trop grande, elle est tronquée. Si l'image est trop petite, par défaut, elle est répétée ligne par ligne, de gauche à droite puis de haut en bas.

La propriété `background-position` permet de préciser le cadrage de l'image en fixant la position du coin en haut à gauche (`top left`) de cette image dans le rectangle du widget.

Exemples : Pour expliquer le fonctionnement du `background`, prenons, cette image de 40×20 pixels :  définie dans le fichier `"monImage.png"` et présentons quelques exemples d'affichage de la division vide suivante définie en HTML par : `<div id="test"></div>`

Voici, en fonction du code CSS présenté à gauche ce qui sera affiché, à droite :

```
#test {
   background-image: url(monImage.png);
   border: 1px black solid;
   whidth: 150px;    height: 50px;
}
```



Par défaut si l'image est trop petite, elle est répétée de gauche à droite puis de bas en haut. Mais on peut empêcher ça avec la propriété `background-repeat` :

```
#test {
   background-image: url(monImage.png);
   background-repeat: no-repeat;
   border: 1px black solid;
   whidth: 150px;    height: 50px;
}
```

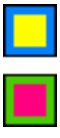
Regardons maintenant comment on peut « découper » une image virtuellement.

Notre fichier `"monImage.png"` contient une image de 40×20 pixels que je veux afficher comme deux image de 20×20 pixels dans deux divisions codées ainsi :

`<div id="testA"></div> <div id="testB"></div>`

Voici le code CSS présenté à gauche et ce qui sera affiché, à droite :

```
#testA {
   background-image: url(monImage.png);
   border: 1px black solid;   whidth: 20px;    height: 20px;
}
#testB {
   background-image: url(monImage.png);
   background-position: -20px 0px;
   border: 1px black solid;   whidth: 20px;    height: 20px;
}
```



On remarque que c'est la même image qui est utilisée, mais décalée de 20 pixels vers la gauche dans le second cas, et comme le widget est de 20×20 pixels, l'image est tronquée.

Ainsi, en utilisant la même image mais en changeant sa position d'origine et en fixant les propriéts `height`) et `height`) il est possible d'effectuer un découpage virtuel de cette image.

Propriétés de transparence (`opacity`) :

— `opacity` fixe le niveau de transparence du widget (valeur : un nombre compris entre 0 et 1 avec : $0 = invisible$ et $1 = opaque$)
exemple : `opacity: 0.5;`

4.2.4 Définition des styles

Passons maintenant à la rédaction de la feuille de style qui va contenir une liste de règles comme celle qu'on vient de voir ci-dessus. On ne va pas donner de solution, mais simplement donner des conseils à suivre.

Dans la rubrique « Page Web » du site ISN une application Web interactive permet d'effectuer des tests simples sur le langage CSS :

http://mathmac.free.fr/PageWeb/AppliCSS/index.html

On rappelle qu'il va falloir modifier en parallèle les deux fichiers `"index.html"` et `"style.css"` et vérifier continuellement.

Pour le titre de la page, placé entre `<h1>` et `</h1>` une règle de la forme : `h1 {...}` sera directement écrite dans le fichier `"style.css"`.

Par contre pour sélectionner le message de mode d'emploi, probablement placé dans une division, il faudra identifier cette division en ajoutant un attribut `id` dans la balise `<div>` et donc modifier d'abord le fichier `"index.html"`.

Par exemple : `<div id="message">`

Ensuite on pourra ajouter au fichier `"style.css"` une nouvelle règle de déclarations de style de la forme : `#message { ... }`

On sait maintenant comment il est possible de découper virtuellement l'image de notre Puzzle. Prenons par exemple une image `"image.png"` de 400×400 pixels que l'on désire découper en 16 morceaux de 100×100 pixels.

On va définir les propriétés de la classe `piece` commune à toutes les pièces.

On va ensuite définir 16 classes nommées de `p0` à `p15`, pour préciser la position d'ancrage de chacun des 16 morceaux d'images.

```
.piece {
   background-image: url(image.png);
   whidth: 100px;
   height: 100px;
}

.p0 { background-position: 0px 0px; }
.p1 { background-position: -100px 0px; }
   ...
.p15 { background-position: -300px -300px; }
```

Ainsi, par exemple, pour visualiser dans une cellule de table le morceau numéro 4 (le premier de la seconde ligne), on aura le code HTML suivant : `<td class="piece p4"></td>`

On remarque que quand l'attribut `class` a pour valeur deux noms de classes séparés par un espace, les propriétés des deux classes sont cumulées.

Comme un clic de l'utilisateur pourra sélectionner une pièce, il faut prévoir un style spécifique pour que la pièce sélectionnée soit visuellement différente de la pièce non sélectionnée, par exemple, pour la pièce ci-dessus : `<td class="pieceSelectionnee p4"></td>`
On ajoutera une règle de la forme : `.pieceSelectionnee {...}` au fichier `"style.css"`, du même type que celle de `.piece` ci-dessus à laquelle on ajoutera, par exemple, une propriété de transparence `opacity`.

Pour procéder à la reconstitution de l'image, des cellules vides seront necessaires sur les grilles de jeu. On pourra définir une règle de calsse de la forme : `.celluleVide { ... }`

On peut aussi définir un style, comme une couleur de fond, qui s'applique à tout le corps du texte avec une règle de la forme : `body { ... }`
ou préciser le style des boutons avec : `button { ... }`

Attention : Ne pas attendre d'avoir terminé le codage pour effectuer des vérifications. Il est impératif, après chaque modification, de sauvegarder les deux fichiers `"index.html"` et `"style.css"`, puis de recharger immédiatement le fichier `"index.html"` dans votre navigateur Internet pour contrôler tout de suite la validité du code modifié.

Vous êtes maintenant satisfait du rendu de votre page Web. Cependant vos styles semblent figés. On va voir maintenant qu'il est possible de modifier l'affichage par programme.

4.3 `"code.js"` un programme codé en JavaScript

On va maintenant créer un nouveau fichier de texte que l'on va nommer, par exemple `"code.js"`. Il va contenir un programme écrit dans le langage JavaScript et qui est interprété par les navigateurs Internet.
On travaillera encore sur deux fichiers en parallèle, `"index.html"` décrit au paragraphe 4.1 page 5 et `"code.js"` notre nouveau fichier.
Avant de commencer la programmation, on va d'abord lier notre page HTML `"index.html"` à notre fichier `"code.js"`.
Pour cela on va ajouter dans l'en-tête de notre document HTML une balise `<script>` avec sa liste attributs-valeurs sous la forme :

```
<script charset="utf-8" type="text/javascript" src="code.js"></script>
```

Les différents attributs sont :

- `charset="utf-8"` indique le mode d'encodage du fichier
- `type="text/javascript"` précise le langage
- `src="code.js"` donne le nom du fichier JavaScript

À l'ouverture du fichier `"index.html"` dans un navigateur Internet les deux fichiers auxiliaires `"style.css"` et `"code.js"` sont aussi ouverts automatiquement et interprétés.

4.3.1 Le langage JavaScript

Le langage JavaScript est un langage de programmation interprété par le navigateur. Il comporte deux aspects : la programmation générale et des fonctions d'accès au DOM. Il permet donc de consulter et de modifier à volonté les propriétés de tous les objets du DOM, et comme

toute modification se répercute immédiatement sur l'affichage dans la fenêtre du navigateur, des changements d'affichage dynamiques sont possibles.

De plus JavaScript permet de prendre en compte les évènements, comme les clics de souris, que l'utilisateur peut provoquer sur la page Web. Ainsi il devient possible de modifier l'aspect graphique de la page en fonction des actions de l'utilisateur.

On ne va pas faire un cours de programmation, mais seulement indiquer comment procéder à partir d'exemples simples. Le projet pourra alors être terminé en s'inspirant de ces exemples.

Attention : Ne pas oublier, après chaque modification, de sauvegarder les trois fichiers et de recharger immédiatement le fichier `"index.html"` dans le navigateur Internet pour contrôler le bon fonctionnement de l'ensemble.

Les outils de développement sont ici indispensables pour cela.

4.3.2 Initialisation du programme

La gestion de ce Puzzle est un programme relativement délicat. On donnera des indications pour orienter la réalisation du projet, sans toutefois donner de réponse à toutes les difficultés.

La première chose à faire est d'initialiser le programme. Pour cela il faut créer une fonction JavaScript, nommée par exemple : `init` et l'appeler dès que la page HTML est chargée.

On va ajouter dans la balise `<body>` du fichier `"index.html"` un nouvel attribut `onload` qui a pour valeur une instruction JavaScript, ici un appel de la fonction d'initialisation :

```
<body onload="init()">
```

Cet appel de fonction sera exécuté après chargement de de la page Web.

Ensuite on définit dans le fichier `"code.js"` la fonction JavaScript :

```
function init() {
  // rien pour l'instant
}
```

Pour l'instant, cette fonction ne fait rien comme l'indique le commentaire. En JavaScript tout ce qui suit `//` jusqu'à la fin de la ligne est un commentaire ignoré à l'exécution.

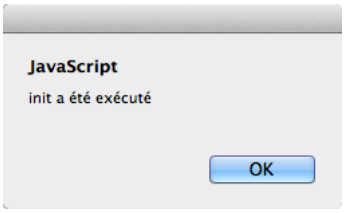
Après enregistrement des deux fichiers `"index.html"` et `"code.js"` on recharge la page dans le navigateur et on utilise la console d'erreur de l'outil de développement pour vérifier que tout s'est bien passé. Même s'il ne s'est rien passé, il y aurait pu y avoir une erreur si le code avait été mal écrit.

4.3.3 Débug : Alerte JavaScript

Voici une petite astuce. Pour être certain que l'initialisation a bien eu lieu, remplacer le commentaire dans la fonction `init` par le code :

```
alert("init a été exécuté");
```

Ceci provoquera, lors du chargement du fichier `index.html`, l'afichage d'un dialogue d'alerte :



L'instruction `alert("message")` peut ainsi être placée dans toute zone de programme chaque fois qu'il est utile de la tester.

4.3.4 Modification d'un objet du DOM

Nous avons prévu dans la page d'accueil *un message de mode d'emploi ou de compte rendu de jeu* qui sera donc modifié en cours de jeu. Nous avons pu créer une division identifiée pour en définir le style, par exemple :

```
<div id="message">Bonjour</div>
```

Pour afficher au départ le mode d'emploi du Puzzle : « Clic sur la pièce sélectionnée puis Clic sur la zone vide où la placer », plutôt que : « Bonjour », il nous faut trouver l'objet du DOM qui représente cette division et en modifier le contenu.

On peut faire ça ainsi dans la fonction `init` :

```
var div_message;

function init() {
   div_message =  document.getElementById("message");
   div_message.innerHTML = "Clic sur la pièce sélectionnée [...] où la placer";
}
```

Il y a maintenant deux instructions dans la fonction `init` :

1. `document` est une *variable globale prédéfinie* de JavaScript qui donne accès au DOM.
   La méthode `getElementById("message")` qui est appelée, retourne l'objet du DOM identifié par le couple attribut-valeur `id="message"` d'une balise HTML.
   L'objet obtenu est affecté à la variable globale `div_message` déclaré avant la fonction en début de programme.
2. en affectant la propriété `innerHTML` de cet objet du DOM `div_message`, le texte html contenu entre les balises `<div>` et `</div>` de la division est modifié ainsi que l'aspect graphique de son widget.

4.3.5 Action du bouton

Le principe est le même que pour l'initialisation, on ajoute cette fois dans la balise `<button>` du fichier `"index.html"` un nouvel attribut `onclick` qui a pour valeur une instruction JavaScript, ici un appel de la fonction `nouvellePartie` :

```
<button onclick="nouvellePartie()">Mélanger les pièces du Puzzle</button>
```

Cet appel de fonction sera exécuté si l'utilisateur clique sur le bouton.

Ensuite on définit dans le fichier `"code.js"` une nouvelle fonction JavaScript :

```
function nouvellePartie() {
  // rien pour l'instant
}
```

Pour coder l'action de ce bouton. il est nécessaire de créer un modèle abstrait du jeu comme nous allons le voir maintenant.

4.3.6 Modélisation du jeu

Pour jouer, il faut pouvoir déplacer facilement, avec la souris, les pièces du Puzzle d'un cellule de grille à l'autre. Il faut aussi pouvoir facilement corriger les erreurs éventuelles.

L'image carrée étant découpée en 16 morceaux, on conseille de prévoir 2 grilles 4×4 comme sur la Figure 2 page 3. La grille de gauche, dont les cellules sont espacées, contiendra au départ les pièces dans le désordre. La grille de droite, dont les cellules sont jointives, contiendra à la fin la solution comme on peut le voir Figure 1 page 2.

Le modèle abstrait du jeu doit décrire l'état du Puzzle. On conseille d'utiliser une table de $2 \times 16 = 32$ cellules. Les 16 premières cellules contiendront les pièces mélangées au départ et les 16 dernières cellules contiendront les pièces ordonnées de la solution.

Nous allons d'abord fixer la valeur de l'attribut `id` et ajouter un attribut `onclick` à chaque cellule `<td>` de la première table associée à la grille de gauche :

```
<td id="C0" onclick="clicCellule(0)"></td>
<td id="C1" onclick="clicCellule(1)"></td>
 ...
<td id="C15" onclick="clicCellule(15)"></td>
```

De même pour la seconde table associée à la grille de droite :

```
<td id="C16" onclick="clicCellule(16)"></td>
<td id="C17 onclick="clicCellule(17)"></td>
 ...
<td id="C31" onclick="clicCellule(31)"></td>
```

Ainsi la fonction `init` va pouvoir récupérer à l'aide de `document.getElementById`, les objets du DOM que notre programme doit contrôler, dans des variables globales. Par exemple `div_message` pour la zone de message et `td_cellule` pour les 32 cellules des grilles de jeu.

Les cellules des grilles sont identifiées par `C0`, `C1` ... `C31`. On peut construire ces noms par programme. Par exemple, la cellule de numéro `x`, aura pour identificateur : `"C" + String(x)`

On peut ainsi récupérer les widgets des cellules dans une variable globale `td_cellule` :

```
td_cellule = [];
for (x = 0; li < 32; x += 1) {
    td_bascule[x] = document.getElementById("C" + String(x));
}
```

Une variable globale supplémentaire `etat_cellule` pourra contenir l'état actuel de chaque cellule, par exemple : le nombre -1 pour une cellule vide, un entier n compris entre 0 et 15 pour représenter le pièce numéro n du Puzzle.

Ne pas confondre le tableau de 32 éléments `td_cellule` des objets ou widgets du DOM représentant les cellules du jeu, avec le tableau de 32 éléments `etat_cellule` représentant par des nombres entiers le contenu de ces cellules.

Ainsi, `td_cellule[3]` référence le widget de la troisième cellule (indexée à partir de 0), en haut à droite de la grille de gauche, et `etat_cellule[3]` caractérise le numéro de la pièce qu'elle contient (ou -1 si elle est vide).

La fonction `init`, pourra ainsi récupérer les éléments du DOM et ensuite initialiser le modèle du jeu puis lancer une nouvelle partie.

```
function init() {
    var x;
    div_message = document.getElementById("message");
    td_cellule = [];
    for (x = 0; li < 32; x += 1) {
        td_cellule[x] = document.getElementById("C" + String(x));
    }
    nouvellePartie();
}
```

La fonction `nouvellePartie` vue au paragraphe précédent va alors placer aléatoirement les morceaux de l'image sur la grille de gauche (cellules de numéro n avec $0 \leqslant n \leqslant 15$) et vider les cellules de la grille solution à droite (cellules de numéro n avec $16 \leqslant n \leqslant 31$) et.

Gestion des cellules : Donnons quelques indications supplémentaires pour vider une cellule ou y placer une pièce du Puzzle en JavaScript.

On pourra relire avec profit le paragraphe 4.2.4 « Définition des styles ».

On peut par exemple vider la cellule numéro x avec le code suivant :

```
etat_cellule[x] = -1;
td_cellule[x].className = "celluleVide";
```

On peut placer dans la cellule numéro x la pièce numéro n de l'image avec le code suivant :

```
etat_cellule[x] = n;
td_cellule[x].className = "piece p" + String(n);
```

On peut sélectionner la cellule numéro x avec le code suivant :

```
td_cellule[x].className = "pieceSelectionnee p" + String(etat_cellule[x]);
```

Ou la dé-sélectionner avec le code suivant :

```
td_cellule[x].className = "piece p" + String(etat_cellule[x]);
```

Mélange aléatoire des pièces du Puzzle : Un mélange de pièces peut être modélisé par une liste de 16 entiers compris entre 0 et 15 dans un ordre aléatoire, comme par exemple :

`[13, 11, 3, 8, 1, 15, 14, 9, 0, 5, 6, 4, 12, 10, 2, 7]`

Le langage JavaScript fournit quelques fonctions mathématiques qui permettent d'obtenir des nombres pseudo-aléatoires :

— `Math.random()` cette fonction retourne nombre décimal compris entre 0 inclus et 1 exclu, et différent à chaque appel

`Math.random()` ⟶ 0.63141422881744832
`Math.random()` ⟶ 0.09417965123429894
`Math.random()` ⟶ 0.70837463880889121

— `Math.floor(x))` retourne le plus grand entier inférieur à x

`Math.floor(0.09417965123429894)` ⟶ 0
`Math.floor(631.041422881744832)` ⟶ 631

On pourra ainsi obtenir des nombres entiers aléatoires compris entre 0 et 15 à l'aide de l'expression suivante : `Math.floor(Math.random() * 16)`

Fin du jeu : Compte tenu du modèle choisi, les morceaux du Puzzle sont placés, au départ, aléatoirement dans les cellules de la table de gauche, numérotées de 0 à 15, et la solution est trouvée si les pièces numérotées de 0 à 15 sont mises dans cet ordre dans les cellules numérotées de 16 à 31, celle de la table de droite.

Jouons enfin : Lorsque l'utilisateur clique sur une cellule la fonction `clicCellule` est appelée, avec pour arguments le numéro de la cellule cliquée. Évidement, pour que le puzzle fonctionne, la fonction `clicCellule` doit modifier l'état des cellules.

Il ne faut pas oublier, non plus, de vérifier à chaque clic si le but est atteint, c'est à dire que l'image est reconstituée. Dans ce cas, il est bon d'afficher un message de victoire !

Il ne reste donc plus qu'à coder cette fonction :

```
function clicCellule(x) {
    // ici on modifie légat des cellules en respectant la règle du jeu
    verifierSiCestFini();
}
```

On ne donnera pas le code du programme à fournir mais seulement quelques précisions utiles sur les évènement que chaque clic peut provoquer.

Le déplacement de chaque pièce du Puzzle se fait en deux temps :

1. on sélectionne une cellule contentant une pièce ou morceau d'image
2. on clique ensuite sur une cellule vide qui va recevoir la pièce sélectionnée

Donc, lors de chaque clic, il faut prendre en compte l'état actuel de la grille, c'est à dire si une pièce est déjà ou non sélectionnée.

Ce problème assez délicat exige d'ajouter au modèle une variable globale supplémentaire `celluleSelectionnee` contentant le numéro de la cellule sélectionnée (entier entre 0 et 31) ou, par exemple, -1 s'il n'y a pas de sélection.

Précisons les différentes situations que doit prendre en compte la fonction `clicCellule` :

- SI une cellule (contentant une image) est déjà sélectionnée
  - SI la cellule cliquée est la cellule sélectionnée
    - ***on dé-sélectionne la cellule***
  - SINON la cellule cliquée n'est pas la cellule sélectionnée
    - SI la cellule cliquée est vide
      - ***la sélection passe de cellule sélectionnée à la cellule cliquée***
- SINON aucune cellule n'est sélectionnée
  - SI la cellule cliquée est vide
    - ***on ne fait rien***
  - SINON la cellule cliquée n'est pas pas vide (elle contient une image)
    - ***la cellule cliquée est sélectionnée***

Débug : Si vous rencontrez des difficultés de mise au point, n'oubliez pas de programmer des alertes de débug comme celle proposée page 17 au paragraphe 4.3.3.

À vos clavier ...

4.3.7 Que faire de plus ?

Votre projet est terminé ... Il ne vous reste plus qu'à jouer et à résoudre le Puzzle !
Ce n'est peut-être pas aussi facile que prévu ...

De nombreuses autres possibilités d'améliorations de l'interface sont possibles en modifiant la page Web `"index.html"` ou la feuille de style `"style.css"`.